SAND2015-3344R

Department of Homeland Security

This page left blank intentionally

TRANSITION TO PRACTICE (TTP)

Test and Evaluation (T&E) Activity 2 Functional Testing Results CodeDNA

VERSION 2.0

U.S. Department of Homeland Security (DHS)

Science and Technology Directorate (S&T) Cyber Division

Document Number: CM/JHU_APL/CodeDNA/A2R/R2.0/2015/006

15 April 2015

Points of Contact:

DHS Sciences and Technologies Transition to Practice Program Manager
Mike Pozmantier

T&E Plan prepared by
Sandia National Laboratories
Steven Hurd

United States Department of Homeland Security

Science and Technology Directorate

Transition to Practice Test and Evaluation Program

Table of Contents

1. Executive Summary

The purpose of this evaluation is to assess the claim made by the technology provider that CodeDNA provides a "reliable, fully automated, fast means for identifying related malware files and linking variants." CodeDNA, unlike most malware variant detectors, is not based on signatures or hashes; it uses instead a lossy compression to transform binaries into a fingerprint that is a string over a small alphabet, and measures the relationship between fingerprints of binaries, returning a relationship score on a scale of 0.0 to 1.00.

The test team tested CodeDNA using four different data sets to assess its ability to recognize variants of PE format software binaries, including benign code and malware, and to group variants into families. The results show that CodeDNA reports related variants with a relatively high level of precision and recall demonstrating CodeDNA's ability to report relevant matches between malware variants. The strongest results were observed using malware samples collected from the wild.

CodeDNA also supports binning the results by relationship scores so that both precision and recall can be adjusted for specific scenarios, to narrow the results (increase the precision) or to broaden the results (increasing the recall). CodeDNA showed itself to be extremely useful in identifying related samples. Thus, adding CodeDNA to an existing suite of tools in an analyst's workflow can allow them to better prioritize and potentially reduce workload, and much more quickly understand relationships and code changes in newly arrived malware.

As is typically the case with newly developed tools, the CodeDNA version tested exhibited some deficiencies, most notably in the user interface that needs to be expanded and improved[1], and the user documentation that needs to be expanded. However, during 6 months of testing, the test team noted no stability issues in using CodeDNA. Not once did CodeDNA fail to run, nor did the machine hosting CodeDNA crash as a result of CodeDNA. This speaks to the maturity of the software, and to support the statement that CodeDNA is ready for pilot test deployment.

[1] See Technology provider's notes in section 4.10

2. Purpose

The purpose of this evaluation is to assess the claim made by the technology provider that CodeDNA provides a “reliable, fully automated, fast means for identifying related malware files and linking variants.” In order to assess this claim the testing team has selected and conducted functional (efficacy and effectiveness) tests based on the use cases anticipated for CodeDNA. One possible use case for CodeDNA is to separate benign binaries from malware when conducting an incident response, or to group similar files based on a threshold. This would reduce the amount of duplicated effort needed by the incident responders. Another possible use case is to use the fingerprint generated by CodeDNA to identify connections between files within families of malware, linking contextual history and behavior information from other sources for malware analysts. In this way CodeDNA can be used in both practitioner and research settings.

3. Test Criteria

3.1 Technology Version

CodeDNA version 1.1 was used for all tests, unless otherwise stated. Note: Some results were obtained using version 1.2, which adds inclusion scoring, a new scoring method that is described and shown in a few examples below.

3.2 Test Hardware

CodeDNA version 1.1 was tested on both a laptop and a desktop. The specifications are provided below.

Laptop:

Dell Latitude e6430
Intel Core i7 @ 2.8Ghz
16GB of DDR3 1600 ram

Desktop:

Dell Optiplex 9800 with a
2.9Ghz Intel Core i7 @ 2.9Ghz
32GB of DDR3 1600 ram

3.3 Test Sets

For the purposes of this evaluation, subsets and complete sets of test binaries were utilized from three malware repositories. The test sets will be described in detail in this section, and in greater detail in the individual results sections for each of the test sets.

3.3.1 Test Set Notes

The Cleanroom Malware Test Set was comprised of 74 binaries after malware with unstated lineage was removed. Additionally, 1000 binaries were used for both the known malware and benignware in the Labeled Malware and Mixed Test Sets (Test Set C and D).

The following table provides information about the test sets (Tests F1 through F6 are defined below):

Type	Description	Quantity	Truth	Test Used In
Well-Characterized (Test Set A)	PE files compiled from open source software repositories. Several versions from each repository will be used.	5 applications each with 4 versions for a total of 20 binaries (benignware).	Full knowledge of the stated origin and of the source code is available. Functionality is well understood.	F1, F2, F3
Cleanroom Malware (Test Set B)	Malware developed *de novo* by MIT Lincoln Laboratory (MITLL) for the DARPA Cyber Genome project	74 malware binaries.	Ground truth of source, design, and functional capabilities are available. (truth tables are discussed further below)	F4
Labeled Malware (Test Set C)	Malware labeled by automated malware analysis system	1000 malware binaries.	Labels generated from dynamic and static analysis	F5
Labeled Mixed (Test Set D)	Malware labeled by automated malware analysis system and benignware	1000 malware binaries and 1000 benignware binaries Total: 2000 binaries	Labels generated from dynamic and static analysis	F6

Software versions of the applications in the Well-Characterized Test Set were carefully identified in order to avoid testing against versions that do not share a majority of the code base, or are too similar in comparison to functionality found in other samples. In order to fulfill this goal, C and C++ source code written for Windows was selected from open source repositories that contained minor bug fixes and functional updates between versions. As CodeDNA only analyzes the executable portions of binaries, the binaries selected were identified as having only one main executable section while also being dynamically linked. This was to help reduce possible issues that might occur from large matching or non-matching non-executable sections. Non-executable sections can create an additional level of confusion if there are embedded text or image files used as indicators.

Additionally, a small subset of samples was taken from the Labeled Malware Test Set/Test Set C. This subset was created using behavioral analysis drawn from a consensus of antivirus products from a subset in which the behavior of the malware was considered well known. This was deemed the best way to ensure full understanding of a small set of ‘wild’ malware, thereby reducing unknowns in the test set, and to increase the perceived reliability in the truth table generated for this test. Further work was done to ensure similarity in the viewable assembly.

3.4 Metrics

The test criteria used were designed to test the stability and accuracy of the scoring provided by CodeDNA. Stability measures whether CodeDNA returns the same results in multiple runs.
Accuracy seeks to measure how well CodeDNA characterizes a set of samples. The following metrics to evaluate the accuracy of scoring were used:

- **Precision** – Fraction of the similarity relationships that are relevant [3], calculated by the following formula: $\frac{TP}{TP+FP}$
- **Recall** – Fraction of relevant relationships that are retrieved [3], calculated by the following formula: $\frac{TP}{TP+FN}$
- **Purity** – Percentage of the most frequently occurring dominant label within a connected component. The dominant label is typically drawn from the ground truth if available. A mean score will be computed across all connected components.
- **Population Margin** - Measure of the difference between the percentage of the most frequently occurring dominant label and the percentage of the second most frequently occurring label within a connected component. A mean score will be computed across all connected components.

The first two metrics (precision and recall) were calculated using truth tables related to similarity matching and the following definitions:

- True Positive (TP) – Similarity match detected by the analysis that exists in the truth table
- False Positive (FP) – Similarity match detected by the analysis that does not exist in the truth table
- False Negative (FN) – Similarity match missed by the analysis that exists in the truth table

Precision and recall are widely used in analyses of searching technologies, but adapting the binary classification of precision and recall to the measure of the degree of relationship provided by CodeDNA required a more nuanced approach. Two factors are important: the relationship score is a range of 0.0 – 1.0 rather than a binary answer, and the minimum score threshold of returned results is set by users. Typically, users will set the threshold at different values depending on what information they are trying to gather. For this reason, precision and recall were tested at varying score thresholds: 0.01, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, and 0.9. It is expected that lower thresholds will return more false matches, and that will affect the accuracy metrics. A higher threshold is more useful in returning relationships with a high degree of confidence, whereas a lower threshold is more useful when trying to find all possible relationships resulting in a higher degree of recall. Reporting precision and recall for a range of thresholds provides a better understanding of how CodeDNA performs across a range of use scenarios.

Purity and population margin were computed based on truth tables created for each of the test sets. These metrics are used in testing the effectiveness of clustering algorithms. Purity and population margin tend to have better scores when groups or clusters are smaller and more homogenous. This is apparent in the results. As the threshold is lowered to include more samples, recall increases and precision decreases. The groups become larger and less homogenous. As a result, purity and population margin scores decline. These results are still included to serve as an additional indicator of correctness in testing, but are not discussed further as the results, as expected, track well with precision and recall and thus do not add information.

In cases where results diverged, additional analysis was conducted as to the reasons for disagreements between the truth sets and the similarity matching score given by CodeDNA. True Positive, False Positive, and False Negative metrics are included in the test script and were also used to help validate the results.

3.5 Truth Tables

Truth tables were generated using what was known about the binaries being tested. Malware, typically classified by its behavior, is named by the first anti-virus provider to publish. The naming may or may not be followed by other anti-virus providers, which makes generating truth tables based on names alone a very dubious and frustrating exercise. Additional indicators, such as the address or method that a malware sample "calls home" to, are used to help identify malware samples with greater accuracy. As little is usually known about the malware authors or operators, this method is treated as the current best practice. The lack of knowledge and varying identification tools and naming schemas make it difficult, if not impossible, to create indisputable ground truth for a set of malware samples for testing a new technology that automates reporting of similarities in the code in two different binaries

In order to address this, the truth table for the Well-Characterized Set (Test Set A) used in the first three tests was created from open source binaries where a great level of detail was known. The source code was small enough to be read and understood, the authors were known, the behavior is known, and the authors provided comments for changes between versions. The behavior between the programs was also understood to vary between samples, but only slightly between versions. Given this information, a test set was created in which source code, assembly, and meta-data were all readily available. These samples were used for test F1 through F3.

Additionally, *de novo* open source malware written specifically for this kind of testing was made available from the MIT Lincoln Laboratory Cyber Genome project, and was used to create the test binaries in the fourth test (F4). This is the set in Test Set B, the Cleanroom Malware set. *De novo*, a Latin expression meaning "afresh" or "from the beginning", was the term used by the Cyber Genome team to show that they had full control in writing the test malware. This test data, the Cleanroom Malware test set, came with a provided truth table. While there are possible issues with this test set, namely a difference between lineage and relationship, this test set was accepted as having known truth as the authors provided detailed comments between versions, diagrams of the intended lineage, and the source code was made available. Further discussion about the differences between the lineage truth tables and relationship truth tables is below.

Truth tables for test set five and six (F5 and F6), the Labeled Malware test set (Test Set C) and Labeled Mixed test set (Test Set D), were drawn from a malware set referred to as the Mannheim set. This test set was generated and grouped by results of behavioral analysis. The Labeled Mixed test set included an additional set of 1,000 benign Windows XP binaries. In this case, the truth tables closely reflected the believed relationships based on behavioral analysis.

3.6 Discussion on Positive & Negative and Relationship

The approach that CodeDNA takes is novel, and while there are tools with similar goals of identifying related malware, the test team is aware of no tools that use the same method for drawing those relationships.

Test cases for tools meant to identify related malware have multiple issues. The test samples are either drawn from the wild, where there is little known but a lot believed about their familial relationships, or from samples written in the lab where there is much known yet little diversity. This creates a problem in there being either quality samples with little known about them, or much known about samples, but an issue with those samples being 'inbred' and of little diversity. In any case, in order to test CodeDNA, attempts were made using both types of samples with varying levels of success in testing.

As such, explaining 'true positive', 'false positive', 'true negative' and 'false negative' is very important. 'Agreement' is the important distinction to be made here. A 'true positive' is when CodeDNA identifies a relationship which is in *agreement* with the truth table (Figure 1).

	Truth Table	CodeDNA
TP	○—○	○—○
TN	○ ○	○ ○
FP	○ ○	○—○
FN	○—○	○ ○

Figure 1 : Test Outcome Explanation

One case noted in these samples is when relationships between malware are loosely defined. This was illustrated in a briefing provided by the technology developer after looking closely at the results from Test Set B, the Cleanroom Malware test set. In this case, CodeDNA identified relationships that the Cleanroom Malware truth table did not identify. Further complicating this is that at varying threshold levels, CodeDNA creates complete pairwise comparisons reporting the percentage of common code between *ALL* samples above a user-defined score threshold whereas a truth table is binary in nature, reporting a relationship whose score threshold is not defined and not knowable.

In addition, in some of the test sets, the idea of lineage versus relationship is introduced. Lineage is a subset of relationship. It can be highlighted in the case of malware where functionality has been added in series to new versions of malware.

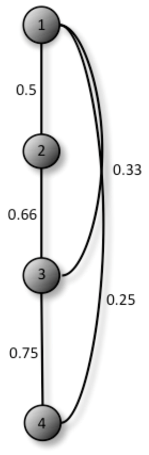


Figure 2: Lineage Graph

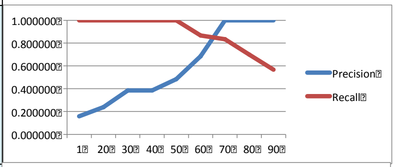
Figure2 is an example of lineage, where a single author has added additional functionality for each new variant. Hypothetically, the assembly changes in set increments, where the differences only increase fractionally. The similarity score between sample 1 and 2 then is ½ or 0.5. As generations are added, the similarity score between generations, given the same quantity of added code for each generation, becomes relatively small. For example, between samples 3 and 4 the score would be 0.75 for ¾. But, when comparing across multiple generations, the similarity scores decrease, such that the final similarity between 1 and 4 is ¼, or 0.25. In the truth table, there would be a relationship where similarity is only 0.25, but there is a known and believed relationship.

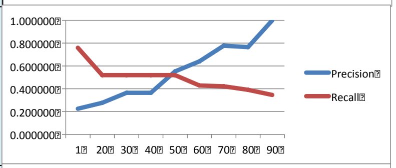
The CodeDNA team recognized this issue, and have made strides towards identifying identical sections in code using 'inclusion scores', available in an updated version of the software. The usage of section scores helps CodeDNA to not only identify similarity scores, but also lineage scores. The Viz tool has demonstrated the capability to highlight and identify this subset of a relationship, and is further shown in the results sections for Test F4.

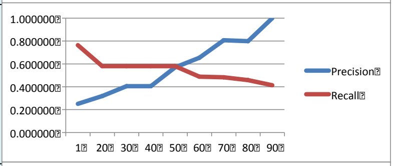
4. Test and Evaluation Results

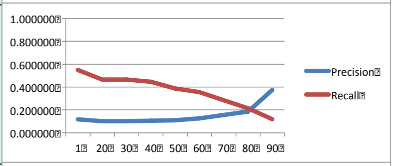
The following figure, Figure 3, is a high-level summary view of the results. The general trend, as was expected, is that as the threshold value decreases precision decreases, while the amount of recall increases. Another way to view this relationship is that as the threshold for what constitutes a similarity relationship is lowered, CodeDNA identifies more related samples, but has a lower confidence that those samples are related. The technology developers anticipate that users will define the scoring threshold differently for different scenarios. A high-speed perimeter defense needs higher precision for automated disposition, and a lower threshold for flagging incoming binaries for decision on the basis of additional metadata, or for referral for more detailed analysis.

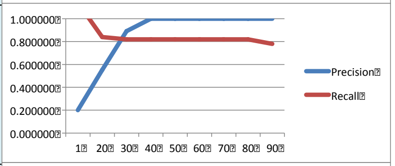
Results File	Truth File	Precision	Recall	Purity	Pop. Margin	Thresh.
'F1_1.csv'	'F1_ground_truth.csv'	0.157895	1.000000	0.200000	0.000000	0.01
'F1_20.csv'	'F1_ground_truth.csv'	0.238095	1.000000	0.625000	0.500000	0.20
'F1_30.csv'	'F1_ground_truth.csv'	0.384615	1.000000	0.777778	0.666667	0.30
'F1_40.csv'	'F1_ground_truth.csv'	0.384615	1.000000	0.777778	0.666667	0.40
'F1_50.csv'	'F1_ground_truth.csv'	0.483871	1.000000	0.777778	0.666667	0.50
'F1_60.csv'	'F1_ground_truth.csv'	0.684211	0.866667	0.850000	0.750000	0.60
'F1_70.csv'	'F1_ground_truth.csv'	1.000000	0.833333	1.000000	1.000000	0.70
'F1_80.csv'	'F1_ground_truth.csv'	1.000000	0.700000	1.000000	1.000000	0.80
'F1_90.csv'	'F1_ground_truth.csv'	1.000000	0.566667	1.000000	1.000000	0.90
'F2_1.csv'	'F2_ground_truth.csv'	0.225446	0.759398	0.614286	0.500000	0.01
'F2_20.csv'	'F2_ground_truth.csv'	0.277108	0.518797	0.675439	0.539474	0.20
'F2_30.csv'	'F2_ground_truth.csv'	0.365079	0.518797	0.760000	0.640000	0.30
'F2_40.csv'	'F2_ground_truth.csv'	0.365079	0.518797	0.760000	0.640000	0.40
'F2_50.csv'	'F2_ground_truth.csv'	0.552000	0.518797	0.827778	0.700000	0.50
'F2_60.csv'	'F2_ground_truth.csv'	0.640449	0.428571	0.837662	0.714286	0.60
'F2_70.csv'	'F2_ground_truth.csv'	0.777778	0.421053	0.944444	0.888889	0.70
'F2_80.csv'	'F2_ground_truth.csv'	0.764706	0.390977	0.954545	0.909091	0.80
'F2_90.csv'	'F2_ground_truth.csv'	1.000000	0.345865	1.000000	1.000000	0.90
'F3_1.csv'	'F3_ground_truth.csv'	0.250188	0.764097	0.631579	0.526316	0.01
'F3_20.csv'	'F3_ground_truth.csv'	0.318182	0.579977	0.691392	0.554945	0.20
'F3_30.csv'	'F3_ground_truth.csv'	0.405145	0.579977	0.771041	0.650679	0.30
'F3_40.csv'	'F3_ground_truth.csv'	0.405145	0.579977	0.771041	0.650679	0.40
'F3_50.csv'	'F3_ground_truth.csv'	0.570136	0.579977	0.837691	0.714597	0.50
'F3_60.csv'	'F3_ground_truth.csv'	0.653313	0.487917	0.848596	0.741148	0.60
'F3_70.csv'	'F3_ground_truth.csv'	0.807322	0.482163	0.950617	0.901235	0.70
'F3_80.csv'	'F3_ground_truth.csv'	0.799599	0.459148	0.959596	0.919192	0.80
'F3_90.csv'	'F3_ground_truth.csv'	1.000000	0.413119	1.000000	1.000000	0.90
'F4_1.csv'	'F4_ground_truth.csv'	0.117389	0.548975	0.961648	0.954545	0.01
'F4_20.csv'	'F4_ground_truth.csv'	0.101190	0.464692	0.972782	0.967742	0.20
'F4_30.csv'	'F4_ground_truth.csv'	0.101190	0.464692	0.972782	0.967742	0.30
'F4_40.csv'	'F4_ground_truth.csv'	0.105946	0.446469	0.972782	0.967742	0.40
'F4_50.csv'	'F4_ground_truth.csv'	0.109114	0.387244	0.972782	0.967742	0.50
'F4_60.csv'	'F4_ground_truth.csv'	0.126214	0.355353	0.942739	0.918300	0.60
'F4_70.csv'	'F4_ground_truth.csv'	0.154895	0.284738	0.942739	0.918300	0.70
'F4_80.csv'	'F4_ground_truth.csv'	0.185771	0.214123	0.906544	0.840372	0.80
'F4_90.csv'	'F4_ground_truth.csv'	0.374101	0.118451	0.936144	0.893121	0.90
'F5_sub_1.csv'	'F5_sub_ground_truth.csv'	0.200704	1.140000	0.604167	0.500000	0.01
'F5_sub_20.csv'	'F5_sub_ground_truth.csv'	0.552632	0.840000	0.902778	0.833333	0.20
'F5_sub_30.csv'	'F5_sub_ground_truth.csv'	0.891304	0.820000	0.979167	0.958333	0.30
'F5_sub_40.csv'	'F5_sub_ground_truth.csv'	1.000000	0.820000	1.000000	1.000000	0.40
'F5_sub_50.csv'	'F5_sub_ground_truth.csv'	1.000000	0.820000	1.000000	1.000000	0.50
'F5_sub_60.csv'	'F5_sub_ground_truth.csv'	1.000000	0.820000	1.000000	1.000000	0.60
'F5_sub_70.csv'	'F5_sub_ground_truth.csv'	1.000000	0.820000	1.000000	1.000000	0.70
'F5_sub_80.csv'	'F5_sub_ground_truth.csv'	1.000000	0.820000	1.000000	1.000000	0.80
'F5_sub_90.csv'	'F5_sub_ground_truth.csv'	1.000000	0.780000	1.000000	1.000000	0.90
'F5_1.csv'	'F5_ground_truth.csv'	0.188347	0.488992	0.997782	0.997168	0.01
'F5_20.csv'	'F5_ground_truth.csv'	0.537820	0.481909	0.998029	0.997390	0.20
'F5_30.csv'	'F5_ground_truth.csv'	0.714576	0.480848	0.998161	0.997515	0.30
'F5_40.csv'	'F5_ground_truth.csv'	0.722272	0.480430	0.997954	0.996375	0.40
'F5_50.csv'	'F5_ground_truth.csv'	0.745770	0.479515	0.997130	0.994260	0.50
'F5_60.csv'	'F5_ground_truth.csv'	0.749120	0.478283	0.997608	0.995217	0.60
'F5_70.csv'	'F5_ground_truth.csv'	0.749619	0.478001	0.997710	0.995419	0.70
'F5_80.csv'	'F5_ground_truth.csv'	0.750377	0.477256	0.997993	0.995986	0.80
'F5_90.csv'	'F5_ground_truth.csv'	0.877329	0.462266	0.998787	0.997573	0.90
'F6_1.csv'	'F6_ground_truth.csv'	0.171280	0.203438	0.999263	0.999094	0.01
'F6_20.csv'	'F6_ground_truth.csv'	0.388323	0.063320	0.991601	0.991498	0.20
'F6_30.csv'	'F6_ground_truth.csv'	0.500244	0.060980	0.994851	0.994743	0.30
'F6_40.csv'	'F6_ground_truth.csv'	0.508791	0.060677	0.995573	0.995041	0.40
'F6_50.csv'	'F6_ground_truth.csv'	0.522335	0.060332	0.995997	0.994867	0.50
'F6_60.csv'	'F6_ground_truth.csv'	0.523850	0.060062	0.995572	0.994409	0.60
'F6_70.csv'	'F6_ground_truth.csv'	0.524050	0.059975	0.994726	0.993576	0.70
'F6_80.csv'	'F6_ground_truth.csv'	0.524430	0.059851	0.994453	0.993133	0.80
'F6_90.csv'	'F6_ground_truth.csv'	0.602087	0.057115	0.994641	0.993516	0.90

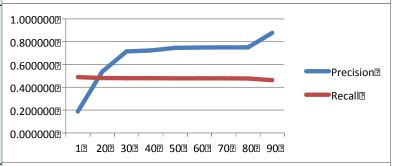












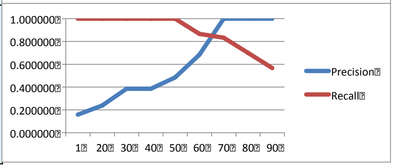


Figure 3: Overall Results Table

4.1 A note on using CodeDNA as a binary classifier

CodeDNA is not solely a binary classifier as it provides a similarity score on a scale of 0.0 to 1.0. Additionally, this score is provided for individual sections of a binary. In order to test CodeDNA, it was considered important to look at precision and recall as a function of some parameter, in this case a similarity score given varying lower bound thresholds for reporting results. Receiver operating characteristic curves (ROC curves) are provided at the beginning of each results section, graphically depicting precision and recall varying over threshold. This report provides information as to how CodeDNA performs in regards to the previously stated test criteria. Note: The maximum similarity score was used for binaries where there were matches on more than one section.

4.2 Recreating Results

The tests conducted were designed to be repeatable for later auditing and correction of any testing errors discovered. There are several scripts written in order to make this easier.

auto_results_analysis.py is used to run all tests on all of the results files and ground truth files used for this round of testing, as this can be error-prone over large test cases. The files have been stored in subfolders for each of the individual tests. The output from auto_results_analysis.py is simultaneously displayed to the screen, as well as recorded to an output file which is hard coded inside the script.

Example use: *python auto_results_analysis.py*

Note: Almost all of the output files required a level of reformatting in order to make the outputs comparable. This was not done to reduce or alter the information provided by CodeDNA in any way, but instead was used for pre-parsing the information for easier data analysis.

4.3 Test F1 - Measure precision, recall, purity, and population margin for Test Set A

Procedure: Using binaries generated from the Well Characterized (Test Set A), CodeDNA was used to identify relationships between binaries. The analysis was conducted using the default settings recommended in the CodeDNA user documentation, with deviations made to vary the threshold value in order to gauge performance at varying levels. The reasons for this variance are given in section 3.3. Precision and recall were calculated from the similarity scores contained in the CSV output from the scoring phase. The scores provided for purity and population margin were calculated via the generated CSV file.

This test set, Test F1 using Test Set A, consisted of 5 applications each having 4 varying versions. The applications were *bgrep, measure, prime, timer, and winutils*. *bgrep* is a grep utility written for Windows which includes the ability to search for binary strings. *measure* is a implementation of parallel-radix sort. *prime* is a utility written to generate prime numbers. *timer* is a C module which implements a simple timer using gettimeofday. *winutils* is an implementation of utilities written to be executed from the windows command window. The binaries tested were selected because they closely fit the criteria selected for testing: they are small enough that the functionality and behavior could be understood, the variances between binaries were limited to bug fixes or small functional updates, and the files were completely composed of executable code. For this iteration the test sets were all compiled with a single compiler with the default optimization flags used.

Results:

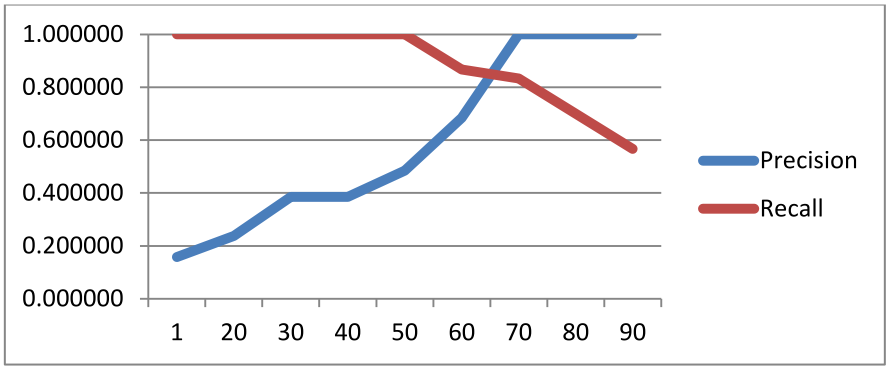


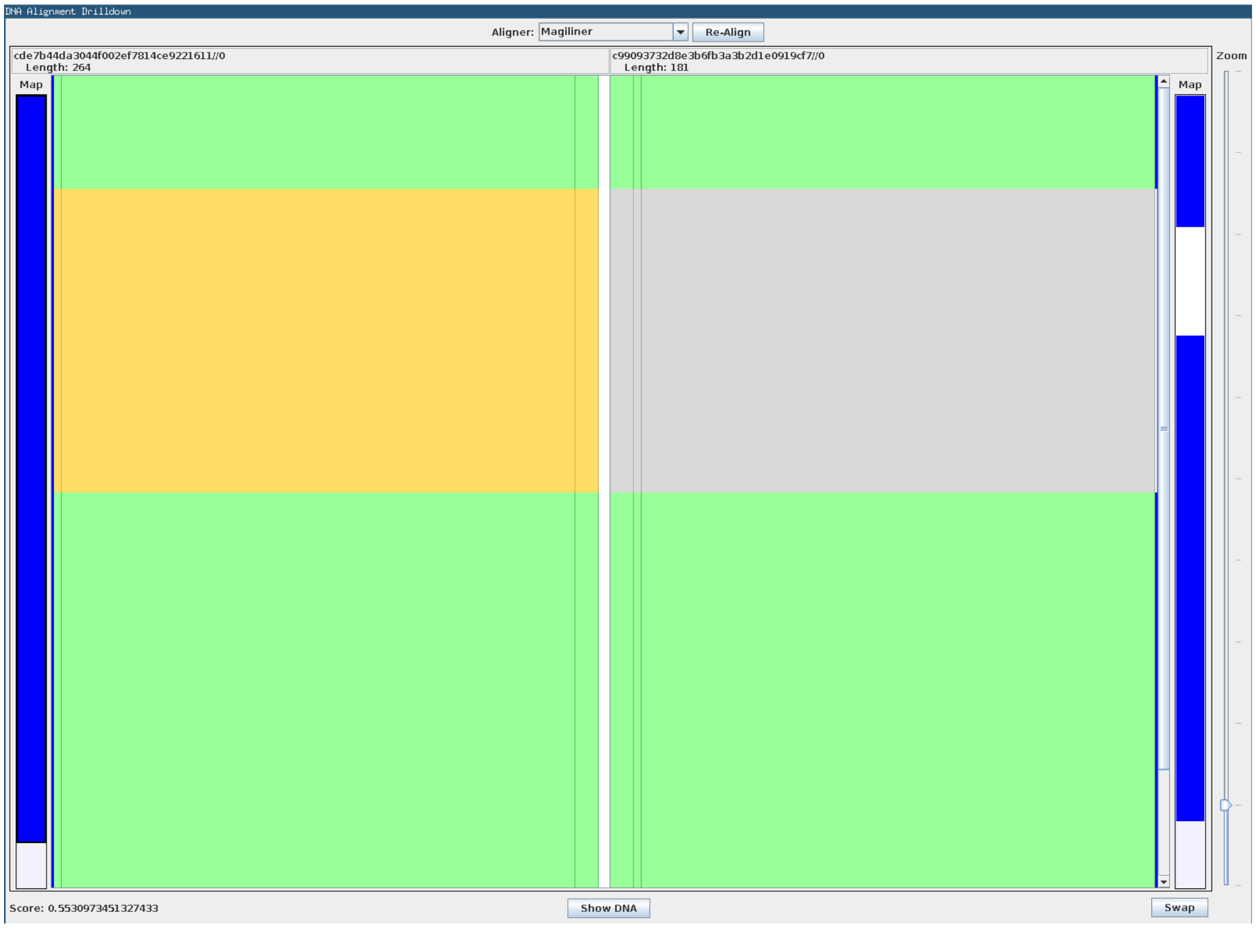
Figure 4: F1 Results

The results from CodeDNA are strong, especially at the high threshold level. The interplay between precision and recall numbers in this case is interesting. In cases where an analyst is looking for a high level of surety in the relationship between files, a higher threshold level would lead to greater confidence in an indicator. In cases where an investigator is more comfortable with lower reliability, but wants to analyze more possible relationships, a lower threshold would yield more possible relationships.

Results Analysis:

F1 is a good place to start with analysis, as it is a smaller set that helps to highlight some of the interesting facets in the results. CodeDNA, when fingerprinting PE format binaries, looks at assembly code, and through a specific vocabulary, is able to create a fingerprint based on what is found. Comparisons between files using CodeDNA's visualization tool, CodeViz, are shown below. What follows are two cases where CodeDNA differs from the truth table that is based on knowledge of the functionality, source code, and history.

As one example, consider the case of *timer2o* and *timer3o*. These are binaries for the second and third version of timer compiled with no optimizations. Drill down visualization results shown in Figure 5 are at a high level; the "zoom lever" provided in drill down lets the user drill down to the detailed content of the file section and review the exact nature of the matching parts and the non-matching parts; the analysis below included detailed examination of the matching and non-matching code.



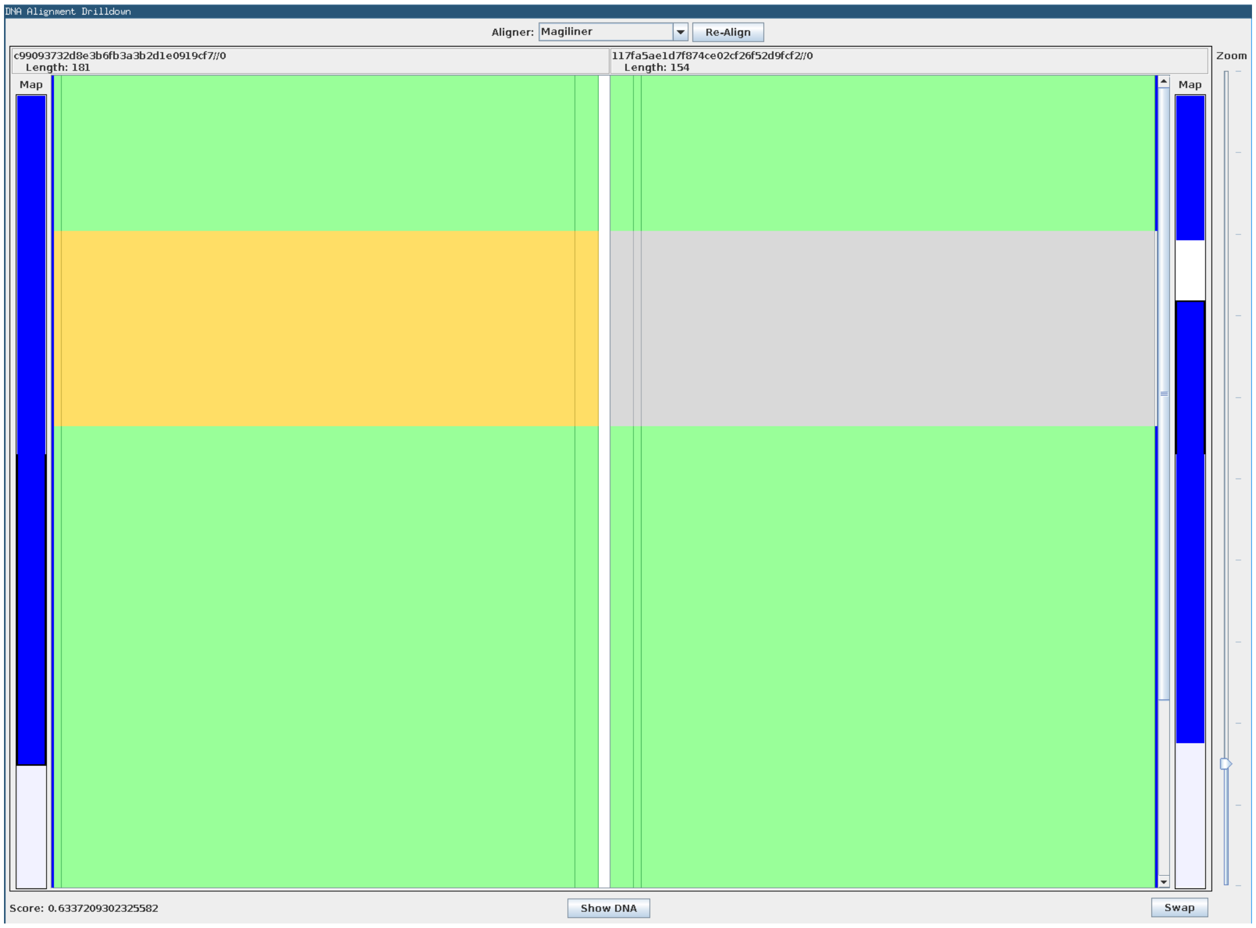
Score: 0.5530973451327433

Figure 5: timer2o vs. timer3o drilldown

In this case the similarity score provided by CodeDNA is given at 0.55. With a threshold above 0.55, this score would be interpreted as a false negative. (i.e., 45% of the code is different, 55% is in common). The green areas on the graph show matching areas, and the yellow and gray area show where assembly exists in one binary that does not in the other. The address space in the binary on the left is linearly increasing; the address space in the binary on the right is re-arranged as needed for matching.

The disassembly tool *objdump* is also useful for verifying the comparison in this case, as the main functions do in fact differ a great deal. Specifically, there is a large amount of code (45%) that exists in one of the binaries and not in the other. In this case, from CodeDNA's measurement of common code, CodeDNA is correct, even though the findings disagree with the known lineage based on authorship, functionality, and sourcing.

timer3o and *winutils3o*, on the other hand, results in a score of 0.63. In this case CodeDNA's result is a false positive, when compared to knowledge about authorship, functionality, and sourcing. Again, looking at the code drill down in CodeDNA is informative. There is a large section of assembly that exists in one binary but not in the other.



c99093732d8e3b6fb3a3b2d1e0919cf7//0
Length: 181

117fa5ae1d7f874ce02cf26f52d9fcf2//0
Length: 154

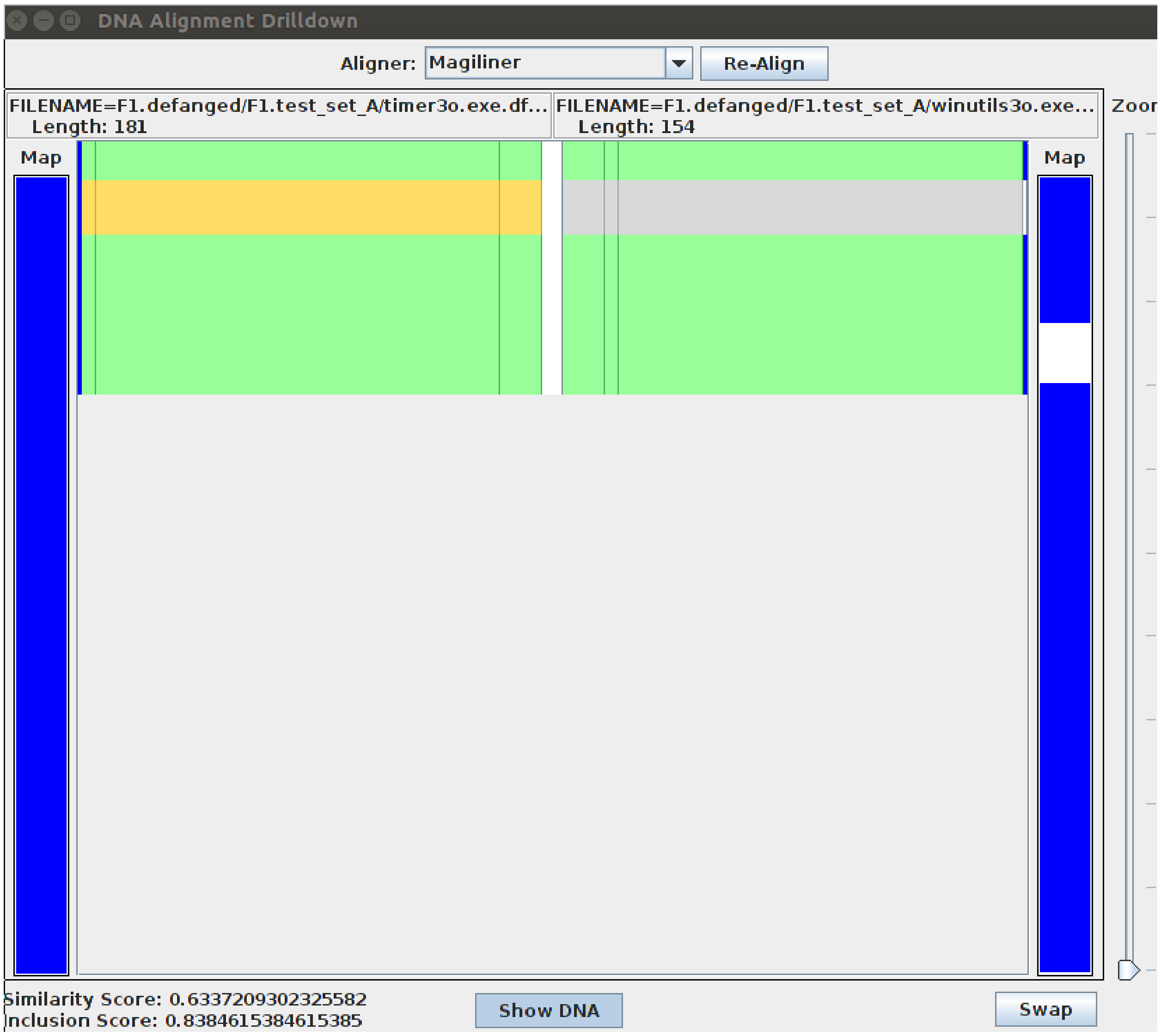
Score: 0.6337209302325582

Figure 6: timer3o vs. winutils3o drilldown

The test team discussed both of these cases with the CodeDNA team. According to the level of assembly code provided in these cases CodeDNA is correctly analyzing the samples. This creates a larger question: Given that there is a large amount of information known about these programs, including the behavior, the authors, the changes between variants, and access to the source code, is it possible that CodeDNA is making a better decision? In this case, given the large amount of information known about the test samples, CodeDNA is not. The scores, however, are close enough to the recommended threshold in this case in that it would be worthwhile to an analyst to conduct further analysis to verify the relationship.

In both these instances it is important to realize that these are extremely short programs. To illustrate this, consider two short programs that are different because one program opens a file and the other prints something to the console. These programs do have different functionality, but a lot of their binary code will be the same at least in part because of “boilerplate” added by the compiler. From the point of view of understanding the functionality it is not correct to say that the programs are related; from the point of view of how much machine code they have in common, it is correct to say that they are related. Notice, however, that CodeDNA’s scores are in a range where it is clear that there is a relationship between these two programs, but it is a weak relationship, and there is a significant percentage of code that does not match.

In order to test this further the following test set includes the same binaries, but adds a second set of binaries compiled from the same source code using a different compiler. Additionally, as CodeDNA is already distinguishing between PE sections, the main program section (the binary section that contains the execution starting address) of the PE may be given additional weight in developing a composite score of all sections.


DNA Alignment Drilldown

Aligner: Magiliner

FILENAME=F1.defanged/F1.test_set_A/timer3o.exe.df...
Length: 181

FILENAME=F1.defanged/F1.test_set_A/winutils3o.exe...
Length: 154

Similarity Score: 0.6337209302325582
Inclusion Score: 0.8384615384615385

Figure 7: timer3o vs. winutils3o drilldown NEW VERSION

Figure 7 shows results of a newer version of CodeDNA that shows the addition of an inclusion score in the lower left hand corner. This score, while not included in the version of CodeDNA provided to Sandia for this round of testing, does help to illustrate improvements made in CodeDNA which both help to highlight the reason for the disagreement between CodeDNA and the generated truth tables, as well as to show efforts made to address this disagreement and improve CodeDNA's detection rate in other cases. Notice also that both of these binaries are extremely small, an order of magnitude smaller than most malware binaries; this results in very small fingerprints, and the potential for less confidence in the results. The weak similarity score and a relatively high inclusion score make it clear that while these two binaries do share a lot of code, one of them has added code that possibly has a different behavior.

4.4 Test F2 - Measure precision, recall, purity and population margin for test set A sources compiled using two different compilers.

Procedure: The same procedure as Test F1 was used but test set A was expanded to include binaries from two different compilers.

Results:

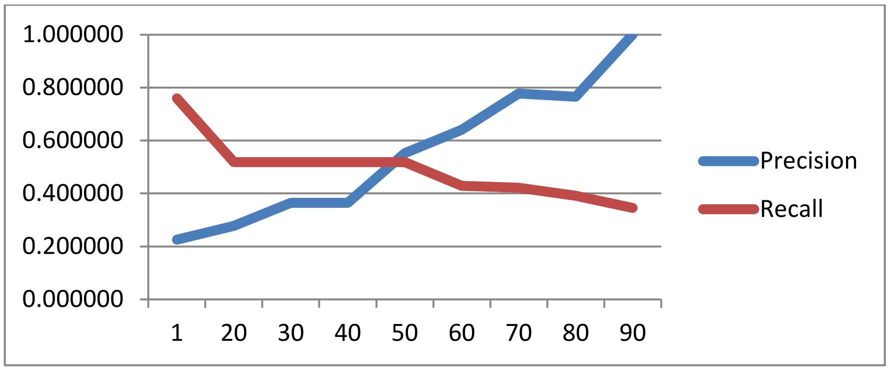


Figure 8: F2 Results

Results analysis: In this case, CodeDNA results helped to confirm part of the issues with using smaller binaries and source files. While this was necessary in order to fully understand the test set, it created a situation where large portions of the binaries tested consisted of 'boiler plate', versus instructions resulting from the source code. The decrease in precision and recall in this case is largely attributed to doubling the number of samples by using a second compiler, and not increasing the size of the sample binaries.

In any case, these results are consistent with expectations, given how CodeDNA analyzes sample files. Despite the decrease in the otherwise very high scores, CodeDNA still appears to be a valuable tool in incident investigation as part of an overall workflow.

Technology provider's note: Because boilerplate code is a known and well-understood issue, the technology developers anticipate options for handling it in a production analysis system. For example, "boilerplate" sections could be tagged with individual fingerprints so that relationships can be evaluated with the boilerplate masked, or the existence of certain classes of boilerplate could be useful in building attribution across disparate pieces of software.

4.5 Test F3 - Measure precision, recall, purity and population margin for test set A sources compiled using two different compilers with varying optimization levels.

Procedure: The same procedure as Test F1 was used but test set A was expanded to include binaries from two different compilers, using multiple levels of compiler optimization. The optimization flags used were default, -2, -3, -fast, and -small.

Results:

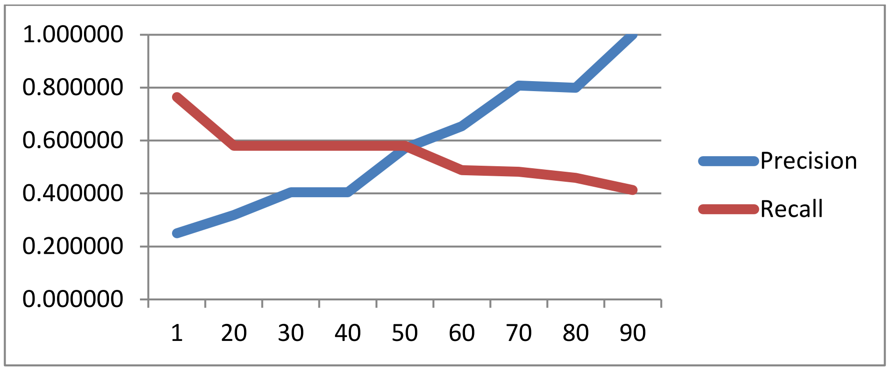


Figure 9: F3 Results

Results analysis: The results from the F3 test set seem to confirm the conclusion from the F2 test set that a change in compiler was the main cause in the lower scores. As more binaries were created by adding compiler optimizations the scores largely stayed the same. The belief in this case is that as the binaries are optimized, the same instructions are used while the code sections are simply moved. This is a case where CodeDNA excels. The results in this case indicate that different optimization flags would not be useful in attempting to avoid detection of relationships by CodeDNA.

4.6 Test F4 - Measure precision, recall, purity and population margin for test set B.

Procedure: The same procedure as Test F1 was used but the test was run on the Cleanroom Malware Set (Test Set B).

Results:

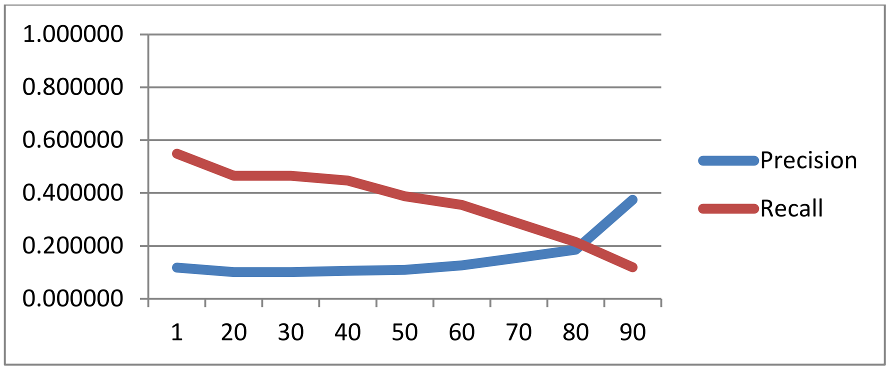


Figure 10: F4 Results

Test set F4 proved to be problematic as a test case. This was not due to any fault with CodeDNA, and the reasons for the test set being problematic were not noted until after testing had completed.

The Cleanroom Malware Set was gathered using the Test Case Composition information provided in the Cyber Genome documentation, largely drawn from the provided Ground Truth Graphs for each labeled family. This set of malware was developed *de novo* by MIT Lincoln Laboratory (MITLL) for the DARPA Cyber Genome Project. Only the cleanroom (notated with a CR) binaries were included from the original test case composition. Further, the S7.MikeBobClusterF.CR.20 was excluded as it stated that there was no lineage for that test set. 74 binaries with known "truth" remained to test.

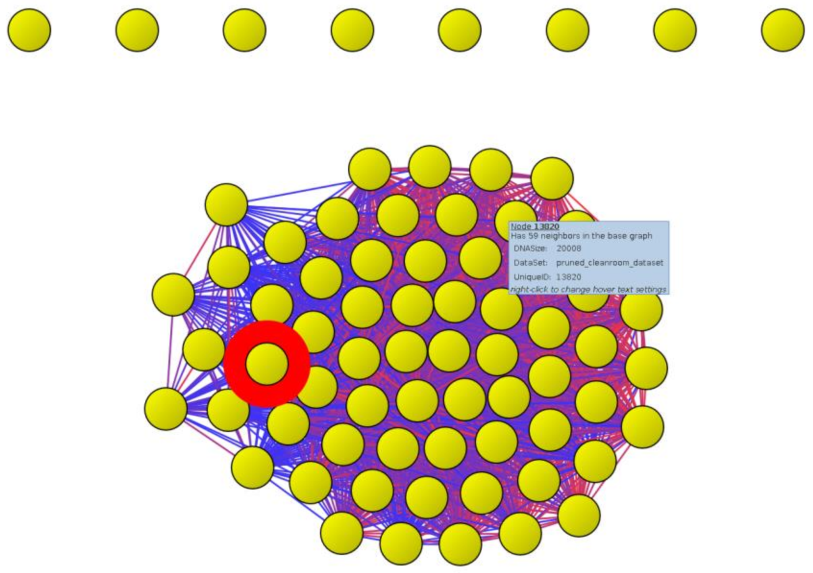
The "truth" provided in this case was problematic in that CodeDNA generates a relationship graph, while the Cleanroom Malware Set provided truth's comprised of Lineage, Clustering, Purpose, Traits, Component Clustering, and Component Lineage. Additionally, not all of these features were provided for all of the families, yet the information was always displayed in a Ground Truth Graph or simple table. This information did not distinguish which feature was being presented. *The variance in the features used to generate the Ground Truth Graphs caused a tremendous impact on CodeDNA's precision and recall scores, and should not be viewed negatively in regards to CodeDNA.*

The CodeDNA scores on the Cleanroom set were lower than anticipated, but after further analysis and comparison with other tools, CodeDNA is shown to have performed admirably. While the numerical scores were found to be accurate, analyzing the meaning of the similarity scores led to interesting insights about the Cleanroom Malware Set. One theory as to why CodeDNA scores seem lower than anticipated is that the Cleanroom test set experienced unforeseen relationships. For example, in the Cyber Genome test set, most of the binaries were written by two authors. As the authors, Mike and Bob, developed the test set, the binaries changed largely due to adding functionality. However, functionality was never removed. In this way, the binaries weren't so much related as they were generational. This process led to an issue where most of the binaries were related by lineage.

If a sample were to be written and then copied you would expect a 100% match. If additional functionality and code were to be added to create a second generation binary, you would expect that comparison between it and the first generation binary would yield a lower percentage match, as the second binary contains instructions not present in the first binary. If a third generation sample were written, then there would be an even lower score between the first generation and the latest generation. This is very likely the case in the Cyber Genome project. CodeDNA measures similarity relationships, and the interpretation of the scores based on different score thresholds will not yield truth tables that exclusively show lineage.

Another factor adds to the difficulty of accurately assessing this test set. In some cases the exact same functionality was added across malware families. Also, as only two authors were used it seems likely that the same coding style is seen by CodeDNA between the stated families. To address this and similar issues, the CodeDNA team stated that they are now able to include an inclusion score, which should help to shed some information as to generational parentage between samples (See Figure 7). If there is a high inclusion score, it might be likely that there is a parental link between samples. This is an area that would be very interesting for additional research; there are tantalizing possibilities for attributing code to individual actors and for tracing relationships across code with very different functionality.

Figure 11 shows the screen capture from CodeDNA Viz of the graph created from the cleanroom malware test set, and helps to show the strong relationship found between all samples provided by the Cyber Genome project, and not just those stated to be familial.



Node 13820
Has 59 neighbors in the base graph
DNASize: 20008
DataSet: pruned_cleanroom_dataset
UniqueID: 13820
right-click to change hover text settings

Figure 11: F4 CodeDNA VizTool

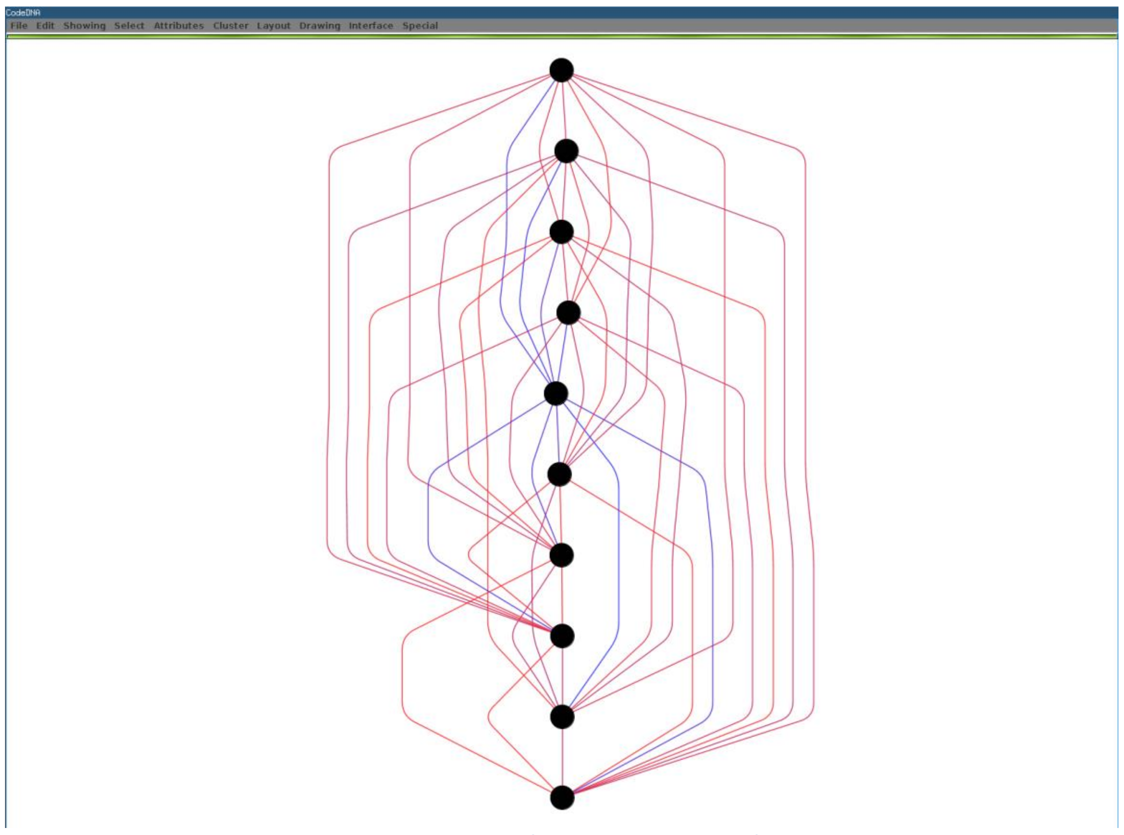

CodeDNA
File Edit Showing Select Attributes Cluster Layout Drawing Interface Special

Figure 12: F4 CodeDNA Lineage Viz Tool

Figure 12 uses the CodeDNA Viz Tool, but instead arranges the samples from the S1.MikeBobStraightLineC2AutoCR10 subset in hierarchy from the source. In this case, the parentage between the malware samples is apparent. The way that that test set was written is clearly shown as functionality increases between the samples. With the samples being generated largely by the same two authors and with the same functionality, the reason for the large cluster seems to be that this test set is largely inbred.

As an additional means of validating the discrepancy with the Cyber Genome test set truth tables, *SSdeep* was used in order to try and identify relationships. *SSdeep* is a popular fuzzy hashing program used to gauge similarities between files. *SSdeep* only identified 4 matches from the entire set, none of which were correct, leading to scores which measured near 0 for both precision and recall. This is not so much a weakness of *ssdeep* as reinforcement of the finding that more work needs to be done to derive lineage from relationships, and that there is interesting potential here for more detailed automation in deriving attribution than is currently possible.

4.7 Test F5 - Measure precision, recall, purity and population margin for test set C.

Procedure: The same procedure as Test F1 was used on the Labeled Malware Set (Test Set C).

Results:

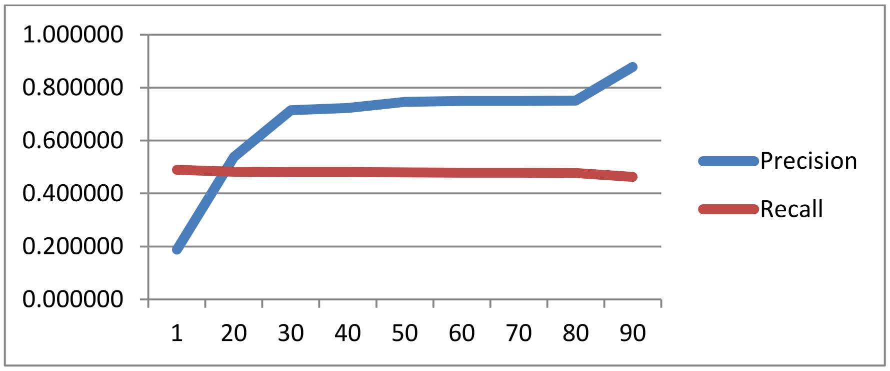


Figure 13: F5 Results

Results analysis: The test set for F5 was selected from the malware repository at Sandia, and was comprised of malware known from the Mannheim Malware set. In this case, malware was identified via behavioral analysis. There is no known large consensus from malware researchers or anti-virus providers regarding this test set as it was largely generated for academic research only.

That said, this set is valuable as it contains samples representative from what is found “in the wild” and has a truth table based on behavioral analysis. CodeDNA was largely in agreement with the truth table results, showing very positive scores.

After examining several of the samples where CodeDNA disagreed with the truth table, it was difficult to assess CodeDNA’s accuracy. In test sets F1 to F3 there is a large amount of information known about the samples. In contrast, test set F4 contained a large amount of information that was believed to be true, but was shown to require greater scrutiny. In the case of F5, there was little consensus among the anti-virus vendors regarding classification of many of the samples. As such, a subset of the malware tested in F5 was selected based on cases where there was stronger sentiment, through consensus by anti-virus providers, largely provided by VirusTotal (see www.virustotal.com), and the belief of a larger degree of variance and confidence in the known functionality. This was the basis for the following test in section 4.8.

4.8 Test F5 Subset – Measure precision, recall, purity and population margin for a subset of Test Set C.

Procedure:

The same procedure as Test F1 was used on a specially selected subset of binaries taken from the Labeled Malware Test Set (Test Set C).

Results:

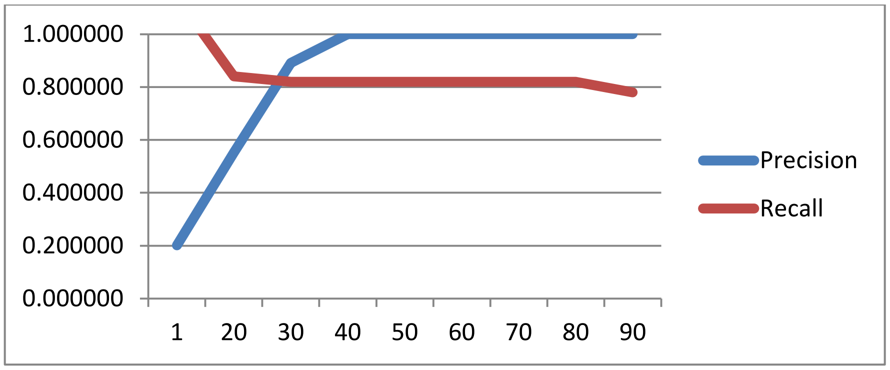


Figure 14: F5 Subset Results

Results analysis: In this case CodeDNA performed well. The test set F5 subset was an effort to improve testing, such that it was not reliant on small binaries, known to have several issues described earlier. As the subset was smaller, we were able to create a truth table with a higher degree of confidence over the larger F5 test set.

In this case CodeDNA maintained a high level of precision with a high and stable level of recall.

4.9 Test F6 - Measure precision, recall, purity and population margin for test set D.

Procedure: The same procedure as Test F1 was used on the Labeled Mixed Set (Test Set D).

Results:

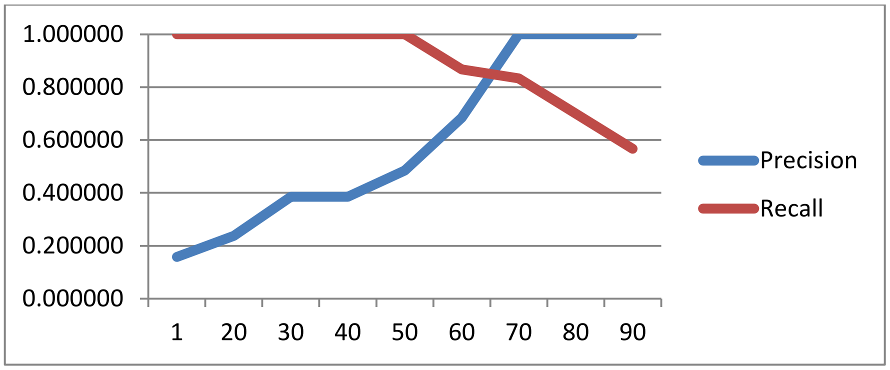


Figure 15: F6 Results

Results analysis: In this case, the addition of Windows XP binaries produced deceiving results. The scores for CodeDNA are high, but have a deceiving drop off point. The results are questionable given that grouping all Windows XP binaries together in a single group caused the truth table to be overly broad, and as such the performance for CodeDNA is likely better than what was shown. It is very likely that CodeDNA identified sub-relationships between the Windows XP binaries which are not reflected in the truth table. Nevertheless, CodeDNA's scores were still high and are in agreement with the results from the larger F5 test set.

4.10 Usability and Testing Notes

During 6 months of testing, the test team noted no stability issues in using CodeDNA. Not once did CodeDNA fail to run, nor did the machine hosting CodeDNA crash as a result of CodeDNA. This speaks to the maturity of the software, and to support the statement that CodeDNA is ready for pilot test deployment.

In testing, two main usability inconveniences were noted. First, when running large test sets, CodeDNA does not adequately notify the tester of completion. The stated practice of watching for the screen to return in the menu for CodeDNA did not always work, and instead the tester had to watch for the results file to stop growing in size. This can be disconcerting in that the tester has no way of knowing if CodeDNA has crashed, or if CodeDNA has completed running.

Second, in testing the Viz Tool, the main frustration came from having the search function for finding specific malware on a separate screen from the main visualization screen. There were ways shown to more adequately arrange and sort the samples, but this was also mildly counter-intuitive. The CodeDNA team has shared methods for showing information in the Viz Tool that would make searching much easier, as well as ways to manipulate the screen to highlight how to better display desired information and relationships between samples.

It was also shown in the following figures, that while the Viz Tool can be very helpful in displaying families, it is possible to overrun the screen depending on the number of samples used. Figure 16 shows a useful mixing of and the interplay between families of malware. Figure 17 shows the screen being overrun as the number of samples increases. Visualization of large amounts of data is a fundamentally challenging problem, and the development team anticipates that any production analysis system will have sophisticated data analysis tools.

Figure 18 helps to show that CodeDNA can be incredibly useful in identifying closely related samples based on a provided threshold. With these families identified an analyst's workload can be decreased to allow better focus on the truly unique or important samples. In this way, analysis after an attack can be triaged in a more meaningful manner.

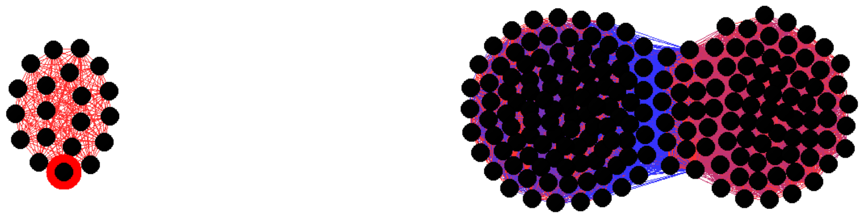


Figure 16: CodeDNA Viz Tool Clusters

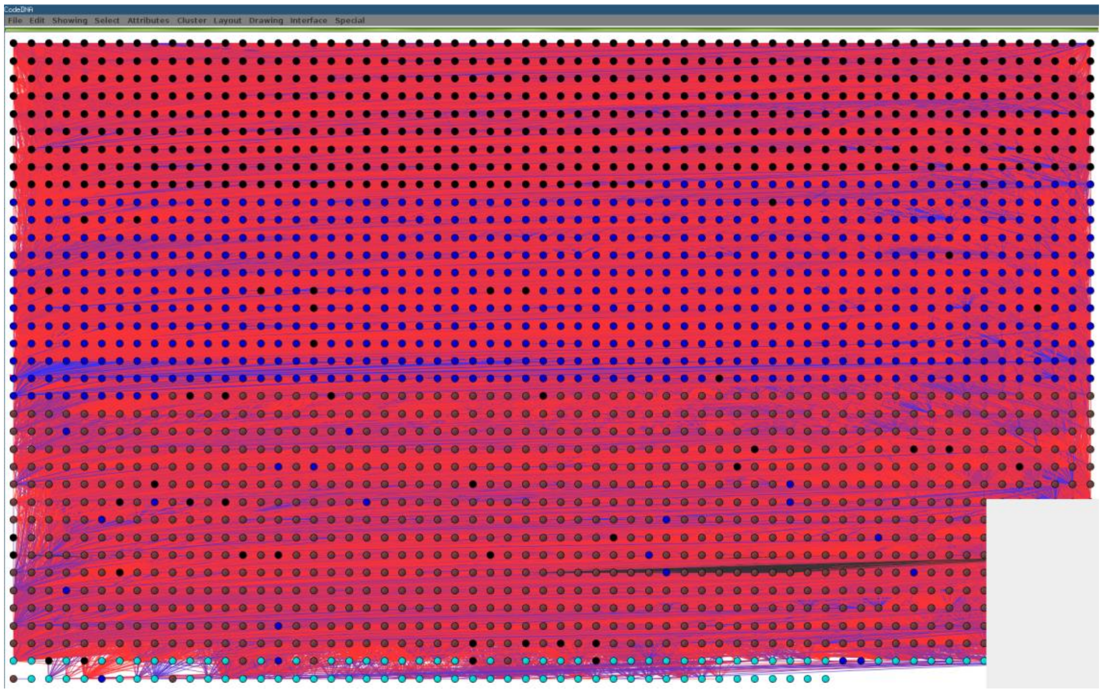

CodeDNA
File Edit Showing Select Attributes Cluster Layout Drawing Interface Special

Figure 17: CodeDNA Viz Tool Screen Overrun

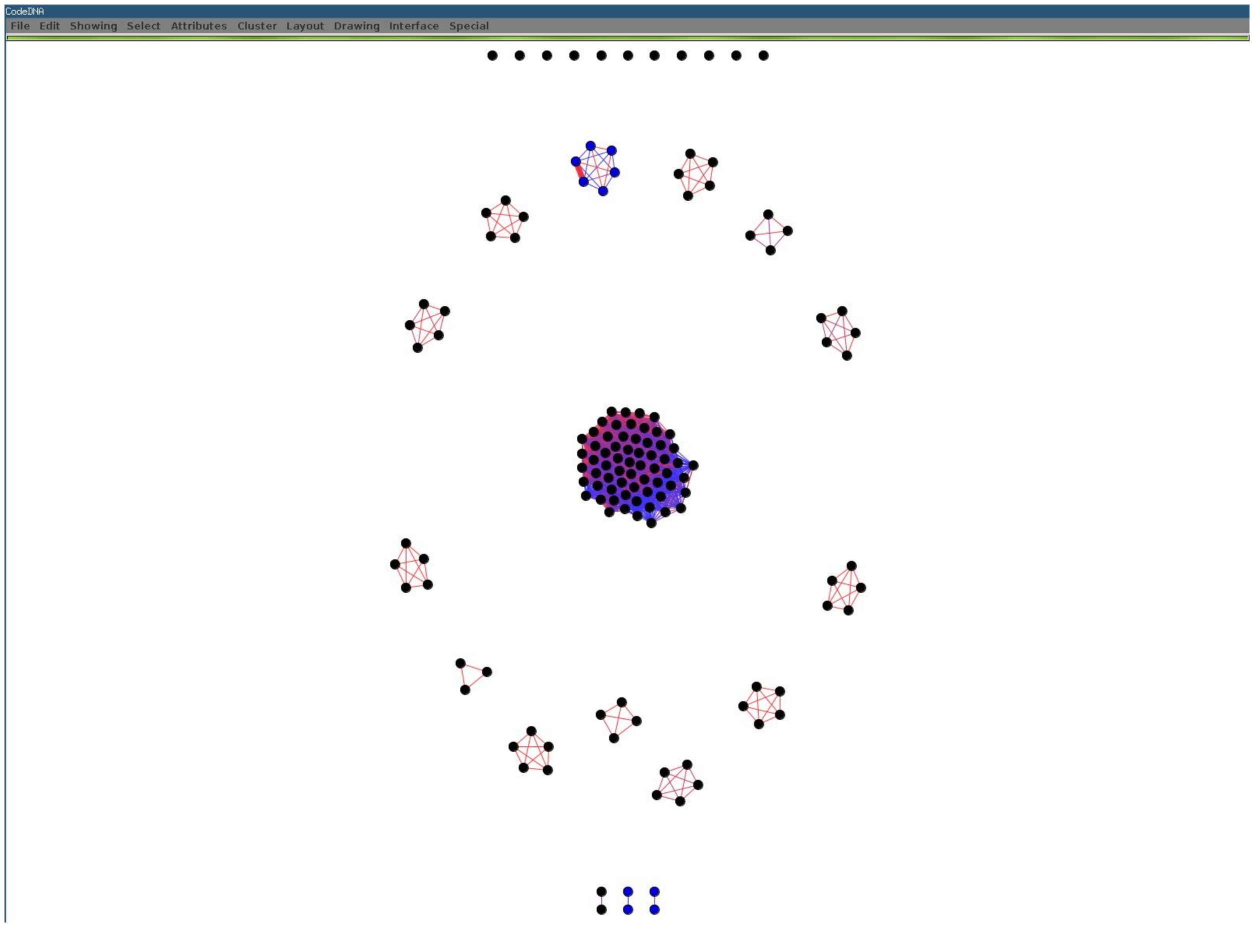

CodeDNA
File Edit Showing Select Attributes Cluster Layout Drawing Interface Special

Figure 18: CodeDNA Triage Helpful Groupings

Many of the recommendations given to the CodeDNA team have already been implemented in subsequent versions, or have improvements in the works. Additional usability notes will be compiled and submitted to the technology provider in regards to improving the documentation, but it should also be stated that while CodeDNA is relatively new, the quality of the documentation provided, the stability of the software, and the level of features clearly demonstrate the level of maturity which CodeDNA has attained, that it is ready for pilot implementation

Technology provider's note: the funding under which CodeDNA was developed was for the core technology only, and did not support the kind of detailed attention that the graphical interface sorely needs. It was always anticipated that the core technology would be embedded into an existing malware processing system that already had its own job management capability and data display capability. Nevertheless, the usability deficiencies pointed out by the testers are real, and were valuable to a team whose focus led to acceptance over time of quirks in the interface and the documentation. There are significant improvements in documentation currently underway. CodeDNA has also now been modified to work as a web service; other API improvements are on the drawing board. There is a lot of interesting work in automating analysis flows and visualizing results that could be done here.

4.11 Conclusion, Recommendation and Future Work

Testing began with the goal of answering the following question. Does CodeDNA provide a “reliable, fully automated, fast means for identifying related malware files and linking variants.” The use cases initially provided were for separating malware from benignware in incident response, and in grouping similar files. Additionally, it was thought that CodeDNA could help to identify families of malware.

CodeDNA was tested using 7 functional tests from 4 distinct test sets. In these tests, CodeDNA reported variants with a high level of precision and recall, while additionally allowing for tuning depending on the situation and the desired return. In cases where the results were not as expected CodeDNA largely highlighted errors in the believed ground truth or in the creation of test samples, while the results pointed to interesting new potential for malware attribution, and directions for further exploration of this aspect. The strongest results came from tests on malware gathered in the wild.

During 6 months of testing, CodeDNA never once crashed a machine, and completed all but the largest test cases in a matter of minutes. CodeDNA also proved to be excellent in grouping related samples, and as a part of an analyst’s toolkit is anticipated to help to reduce the amount of duplicated effort by eliminating the need to conduct detailed analysis of similar samples. Further, CodeDNA can aid in prioritizing workload as well as quickly highlighting believed relationships between samples seen previously as well as newly-arrived malware.

As is expected with any new technology, CodeDNA exhibited some deficiencies in the user interface and documentation, and some improvements were found which will hopefully make CodeDNA a more complete tool. However, given CodeDNA’s stability and performance, the test team believes that CodeDNA is ready for deployment in a pilot test implementation.

5. References

[1] BinDiff, http://www.zynamics.com/bindiff.html
[2] bsdiff, http://www.daemonology.net/bsdiff/
[3] Precision and recall, Wikipedia, http://en.wikipedia.org/wiki/Precision_and_recall
[4] Meila, M., Comparing Clusterings by the Variation of Information, http://www.stat.washington.edu/mmp/Papers/jasa-compare.ps
[5] SSdeep, http://SSdeep.sourceforge.net/
[6] Van Randwyk, J.; Ken Chiang; Lloyd, L.; Vanderveen, K., "Farm: An automated malware analysis environment," Security Technology, 2008. ICCST 2008. 42nd Annual IEEE International Carnahan Conference on , vol., no., pp.321,325, 13-16 Oct. 2008

6. Appendix

6.1 Results Spreadsheet

Results File	Truth File	Precision	Recall	Purity	Pop. Margin	Thresh.
'F1_1.csv'	'F1_ground_truth.csv'	0.157895	1.000000	0.200000	0.000000	1
'F1_20.csv'	'F1_ground_truth.csv'	0.238095	1.000000	0.625000	0.500000	20
'F1_30.csv'	'F1_ground_truth.csv'	0.384615	1.000000	0.777778	0.666667	30
'F1_40.csv'	'F1_ground_truth.csv'	0.384615	1.000000	0.777778	0.666667	40
'F1_50.csv'	'F1_ground_truth.csv'	0.483871	1.000000	0.777778	0.666667	50
'F1_60.csv'	'F1_ground_truth.csv'	0.684211	0.866667	0.850000	0.750000	60
'F1_70.csv'	'F1_ground_truth.csv'	1.000000	0.833333	1.000000	1.000000	70
'F1_80.csv'	'F1_ground_truth.csv'	1.000000	0.700000	1.000000	1.000000	80
'F1_90.csv'	'F1_ground_truth.csv'	1.000000	0.566667	1.000000	1.000000	90
'F2_1.csv'	'F2_ground_truth.csv'	0.225446	0.759398	0.614286	0.500000	1
'F2_20.csv'	'F2_ground_truth.csv'	0.277108	0.518797	0.675439	0.539474	20
'F2_30.csv'	'F2_ground_truth.csv'	0.365079	0.518797	0.760000	0.640000	30
'F2_40.csv'	'F2_ground_truth.csv'	0.365079	0.518797	0.760000	0.640000	40
'F2_50.csv'	'F2_ground_truth.csv'	0.552000	0.518797	0.827778	0.700000	50
'F2_60.csv'	'F2_ground_truth.csv'	0.640449	0.428571	0.837662	0.714286	60
'F2_70.csv'	'F2_ground_truth.csv'	0.777778	0.421053	0.944444	0.888889	70
'F2_80.csv'	'F2_ground_truth.csv'	0.764706	0.390977	0.954545	0.909091	80
'F2_90.csv'	'F2_ground_truth.csv'	1.000000	0.345865	1.000000	1.000000	90
'F3_1.csv'	'F3_ground_truth.csv'	0.250188	0.764097	0.631579	0.526316	1
'F3_20.csv'	'F3_ground_truth.csv'	0.318182	0.579977	0.691392	0.554945	20
'F3_30.csv'	'F3_ground_truth.csv'	0.405145	0.579977	0.771041	0.650679	30
'F3_40.csv'	'F3_ground_truth.csv'	0.405145	0.579977	0.771041	0.650679	40
'F3_50.csv'	'F3_ground_truth.csv'	0.570136	0.579977	0.837691	0.714597	50
'F3_60.csv'	'F3_ground_truth.csv'	0.653313	0.487917	0.848596	0.741148	60
'F3_70.csv'	'F3_ground_truth.csv'	0.807322	0.482163	0.950617	0.901235	70
'F3_80.csv'	'F3_ground_truth.csv'	0.799599	0.459148	0.959596	0.919192	80
'F3_90.csv'	'F3_ground_truth.csv'	1.000000	0.413119	1.000000	1.000000	90
'F4_1.csv'	'F4_ground_truth.csv'	0.117389	0.548975	0.961648	0.954545	1
'F4_20.csv'	'F4_ground_truth.csv'	0.101190	0.464692	0.972782	0.967742	20
'F4_30.csv'	'F4_ground_truth.csv'	0.101190	0.464692	0.972782	0.967742	30
'F4_40.csv'	'F4_ground_truth.csv'	0.105946	0.446469	0.972782	0.967742	40
'F4_50.csv'	'F4_ground_truth.csv'	0.109114	0.387244	0.972782	0.967742	50
'F4_60.csv'	'F4_ground_truth.csv'	0.126214	0.355353	0.942739	0.918300	60
'F4_70.csv'	'F4_ground_truth.csv'	0.154895	0.284738	0.942739	0.918300	70

'F4_80.csv'	'F4_ground_truth.csv'	0.185771	0.214123	0.906544	0.840372	80
'F4_90.csv'	'F4_ground_truth.csv'	0.374101	0.118451	0.936144	0.893121	90
'F5_sub_1.csv'	'F5_sub_ground_truth.csv'	0.200704	1.140000	0.604167	0.500000	1
'F5_sub_20.csv'	'F5_sub_ground_truth.csv'	0.552632	0.840000	0.902778	0.833333	20
'F5_sub_30.csv'	'F5_sub_ground_truth.csv'	0.891304	0.820000	0.979167	0.958333	30
'F5_sub_40.csv'	'F5_sub_ground_truth.csv'	1.000000	0.820000	1.000000	1.000000	40
'F5_sub_50.csv'	'F5_sub_ground_truth.csv'	1.000000	0.820000	1.000000	1.000000	50
'F5_sub_60.csv'	'F5_sub_ground_truth.csv'	1.000000	0.820000	1.000000	1.000000	60
'F5_sub_70.csv'	'F5_sub_ground_truth.csv'	1.000000	0.820000	1.000000	1.000000	70
'F5_sub_80.csv'	'F5_sub_ground_truth.csv'	1.000000	0.820000	1.000000	1.000000	80
'F5_sub_90.csv'	'F5_sub_ground_truth.csv'	1.000000	0.780000	1.000000	1.000000	90
'F5_1.csv'	'F5_ground_truth.csv'	0.188347	0.488992	0.997782	0.997168	1
'F5_20.csv'	'F5_ground_truth.csv'	0.537820	0.481909	0.998029	0.997390	20
'F5_30.csv'	'F5_ground_truth.csv'	0.714576	0.480848	0.998161	0.997515	30
'F5_40.csv'	'F5_ground_truth.csv'	0.722272	0.480430	0.997954	0.996375	40
'F5_50.csv'	'F5_ground_truth.csv'	0.745770	0.479515	0.997130	0.994260	50
'F5_60.csv'	'F5_ground_truth.csv'	0.749120	0.478283	0.997608	0.995217	60
'F5_70.csv'	'F5_ground_truth.csv'	0.749619	0.478001	0.997710	0.995419	70
'F5_80.csv'	'F5_ground_truth.csv'	0.750377	0.477256	0.997993	0.995986	80
'F5_90.csv'	'F5_ground_truth.csv'	0.877329	0.462266	0.998787	0.997573	90
'F6_1.csv'	'F6_ground_truth.csv'	0.171280	0.203438	0.999263	0.999094	1
'F6_20.csv'	'F6_ground_truth.csv'	0.388323	0.063320	0.991601	0.991498	20
'F6_30.csv'	'F6_ground_truth.csv'	0.500244	0.060980	0.994851	0.994743	30
'F6_40.csv'	'F6_ground_truth.csv'	0.508791	0.060677	0.995573	0.995041	40
'F6_50.csv'	'F6_ground_truth.csv'	0.522335	0.060332	0.995997	0.994867	50
'F6_60.csv'	'F6_ground_truth.csv'	0.523850	0.060062	0.995572	0.994409	60
'F6_70.csv'	'F6_ground_truth.csv'	0.524050	0.059975	0.994726	0.993576	70
'F6_80.csv'	'F6_ground_truth.csv'	0.524430	0.059851	0.994453	0.993133	80
'F6_90.csv'	'F6_ground_truth.csv'	0.602087	0.057115	0.994641	0.993516	90